REMARKS

This Reply is in response to the Office Action of December 21, 2006 in connection with the above-identified patent application. Reconsideration of this application in view of the following remarks is respectfully requested.

The Drawings

The informal drawings that were originally filed with this patent application were rejected as not being in compliance with 37 C.F.R. §1.84(g),(l), and (p). Accordingly, applicants are submitting formal drawings herewith that are fully in compliance with the requirements of 37 C.F.R. §1.84.

The Claims

Claims 1-10 and 15-18 were rejected under 35 U.S.C. §102(a) as being anticipated by the Smatch C source checker ("Smatch") as evidenced by the documents Installing Smatch!!! ("SmInst"), Smatch!!! ("SmMain"), Smatch Intermediate Code Representation!!! ("SmIR"), Using Smatch!!! ("SmUsing"), and Using Smatch.pm!!! ("SmUpm"). Claims 11-14 were rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,960,202 (Granston et al.) and the Smatch documents. These rejections are respectfully traversed.

Applicants' invention relates to ways to perform static error analysis in complex software environments.

As applicants explain in their specification, static error analysis tools operate on code that is not running during the analysis process. Although static error analysis tools make certain simplifying assumptions, static analysis tools can detect errors that are often missed when using dynamic analysis tools alone. As an example, static analysis may detect an illegal operation that is contained in a rarely traversed or otherwise hard-to-test conditional branch path.

Although it is desirable to use static error analysis tools, there are obstacles to performing static error analysis in modern software development environments. For example, in many modern software development environments, source code, compilers, and ancillary software components are distributed across various different directories and systems.

As a result of this complexity, software developers typically use build management utilities such as the "Make" program to assist in the process of building executable code. Software developers also construct their own build management utilities (e.g., by writing customized shell scripts). As the software development environment at a given organization matures, it is commonplace for the overall software build process used by that organization to involve many directory

changes, variable assignments, calls of different build management utilities, etc.

The use of static analysis debugging tools in this type of complex build environment is difficult. Static analysis software tools operate on source code, so all of the relevant source code to be debugged must be identified. Different static analysis tools may be used in place of different compilers, so it may be necessary to identify which compilers are normally used to compile which source code files during the build process. Although it may be possible for the proper static analysis tools to be selected and invoked manually, this process can be time consuming and prone to errors, particularly when it is necessary to set up and run a new suite of static analysis tools in a relatively short period of time in a complex build environment.

To address these problems with the conventional deployment of static error analysis tools, applicants have developed improved techniques for using static analysis tools to perform static error analysis on source code files.

There are four independent claims in applicant's patent application (claims 1, 7, 11, and 15), each of which is directed to a different technique for using static analysis tools to perform static error analysis.

Claims 1-6

With the arrangement of independent claim 1, a build program is run on a computer system to invoke compilers on the computer system that compile source code files into executable code. The process of running the build program produces a build program output. As specified by claim 1, a static analysis tool management program is run on the computer system to invoke the static analysis tools. This produces corresponding static error analysis results. Claim 1 requires that the static analysis tool management program accept the source code and the build program output as inputs.

Nothing like this arrangement is shown or suggested by the Smatch documentation.

As described on page 1 of SmMain, Smatch uses a modified gcc compiler to generate .c.sm files. The .c.sm files are analyzed by Smatch scripts that print out error messages. The modified gcc compiler accepts source code as an input and produces .c.sm files (and executable code) as preprocessed output. Smatch's checker scripts process the preprocessed output from the modified gcc compiler to produce static analysis results.

In the Office Action, the Examiner suggests that Smatch's modified gcc compiler is that same as a build program. This is incorrect. Build programs such as the Make program are

programs that invoke appropriate compilers to compile source code files. The source code files may be in different languages, so the build program often invokes multiple compilers suitable for compiling source code files in multiple languages. Build programs may also call a number of build management utility programs. Build programs do not perform the functions of a compiler nor do compilers perform the functions of build programs. Build programs and compilers are simply two different types of programs.

Build programs are programs that invoke compilers rather than being compilers themselves, so it is not correct to equate Smatch's modified gcc compiler with applicants' claimed build program as urged in the Office Action. Claim 1 explicitly requires that the build program invoke compilers on the computer system that compile source code files into executable code. Because the modified gcc compiler in Smatch does not invoke a single compiler, let alone invoke "compilers" as required by claim 1, the modified gcc compiler in Smatch does not meet the requirements of claim 1.

Smatch also fails to show or suggest "running a static analysis tool management program to invoke the static analysis tools and produce corresponding static error analysis results" as required by claim 1. In the Office Action, it was suggested that one of the Smatch checker scripts could be considered to be

the same as applicants' claimed static analysis tool management program. Applicants disagree.

As described in SmMain, the .c.sm files are "piped through individual Smatch scripts." These scripts are not called by a static analysis tool management program. Rather, an individual types in a command to invoke a desired script. This is set forth in the first paragraph of SmIR, which explains how one types "./checker_script.pl file.c.sm" to invoke a desired checker script. The checker scripts in Smatch are therefore manually invoked as needed by a user, rather than being invoked by a static analysis tool management program. Moreover, each checker script is not a separate static analysis tool management program, because the checker scripts process the .sm files, they do not "invoke static analysis tools" as required by claim 1.

Claim 1 is therefore directed toward a method that involves running a build program that invokes compilers, whereas the Smatch modified gcc compiler is not a build program and does not invoke compilers. Claim 1 is also directed toward a method that runs a static analysis tool management program to invoke static analysis tools, whereas Smatch's manually invoked checker scripts are not static analysis tool management programs and do not invoke static analysis tools. For at least these reasons, the Smatch documents do not disclose the features of claim 1. Claim 1 is therefore not anticipated by the Smatch documents and

is in condition for allowance. Claims 2-6 depend from claim 1 and are allowable because claim 1 is allowable.

Claims 7-10

Claim 7 is directed toward a method for using static analysis tools on a computer system to perform error analysis on source code files. The source code files may be compiled on the computer system using compilers having corresponding compiler names that are invoked using a build program during a build process. The method of claim 7 involves creating a new directory on the computer system. The method of claim 7 also involves modifying the search path on the computer system so that the new directory is included first. The static analysis tools are placed into the new directory and are given names matching the compiler names. Static analysis is performed by running the build program so that the static analysis tools with the names matching the compiler names are invoked.

Smatch uses a different approach. In the Smatch arrangement, the original compiler is replaced by modified compiler. The modified compiler is invoked in place of the original compiler. To ensure that the modified compiler is invoked properly, the modified compiler is given a different name than the original compiler. The name of the original compiler is gcc. The name of the modified compiler is xgcc. In

the user's makefile, the user comments out the original variable cc using the comment symbol "#" and inserts a new variable that refers to the modified compiler name xgcc. This process is described in step 1a) of SmUsing.

With the Smatch approach, there is no need to modify the search path as required by claim 7, because the modified compiler is called using the variable cc. Smatch therefore does not show or suggest modifying a search path on a computer system so that a new directory is included first in a search path as required by claim 7.

Moreover, with the Smatch approach, the modified compiler that is called has a different name than the original compiler. The name of the modified compiler is xgcc, whereas the name of the original compiler is gcc. With the method of claim 7, the static analysis tools are placed into a new directory and are given names matching the compiler names. Static analysis is then performed by running a build program so that the static analysis tools with the names matching the compiler names are invoked. Unlike the Smatch approach, applicants' claimed method uses static analysis tools with names that match the compiler names. In the Office Action, it is suggested that the modified compiler xgcc in Smatch is a static analysis tool having a name that matches a compiler name. This

is incorrect, since the name xgcc is different than the compiler name gcc.

Because claim 7 requires modification of a search path so that a new directory is included first in a search path, whereas Smatch makes no search path modifications, claim 7 is not anticipated by Smatch. Claim 7 is also not anticipated by Smatch because claim 7 requires static analysis tools having names that match compiler names, whereas no such matching names are present in Smatch.

For at least these two reasons, claim 7 is not anticipated by Smatch. Claim 7 is therefore patentable, as are claims 8-10, which depend from claim 7.

Claims 11-14

Claim 11 is directed toward a method for using static analysis tools on a computer system to perform error analysis. With the method of claim 11, error analysis is performed on source code files that may be compiled on the computer system using compilers that are invoked using a build program during a build process. The computer system has an operating system. The method of claim 11 involves running the build program on the computer system. The method of claim 11 also involves running a monitoring program on the computer system while the build program is running to compile the source code files. The

monitoring program monitors activity between the build program and the operating system. The output from the monitoring program is used to run the build program with the static analysis tools substituted for the compilers so that the static analysis tools perform static error analysis on the source code files.

The steps involved in using this technique to perform static analysis are shown in FIG. 7 of applicants' patent application. As shown in FIG. 7, the monitoring program (also sometimes referred to as a trace program) is started at step 98. At step 100, the build program is run while using the trace program to monitor activity between the build program and the operating system. The information captured by the trace program is used to run a version of the build program with static analysis tools substituted for compilers (step 104).

In the Office Action, claim 11 was rejected under 35 U.S.C. §103(a) based on a combination of the Granston and Smatch references. In particular, it was suggested that Granston discloses applicants' claimed monitoring program and that Smatch teaches using the output from such a monitoring program to run a build program with static analysis tools substituted for compilers.

However, Granston does not disclose a monitoring program that produces an output suitable to run a build program

as required by claim 11. Rather, Granston discloses a wrapper program that is stored in memory where the compiler usually resides (see column 3, lines 51-54). When the user runs a build program, the build program attempts to invoke the compiler (column 3, lines 59-61). Because the wrapper program is stored where the compiler program is normally stored, the wrapper program is invoked in place of the compiler (column 3, lines 61-66). The wrapper program receives the compiler command line and determines which compiler options are contained in the command line (column 3, line 66 to column 4, line 1). The wrapper program then generates a log file that lists compiler options from the command line (column 4, lines 1-3). After generating the log file, the wrapper program invokes the compiler (column 4, lines 12-14).

In the Office Action, it was conceded that Granston fails to disclose using output from a monitoring program to run a build program with static analysis tools substituted for compilers so that static analysis tools perform static error analysis on the source code files. It was suggested that Smatch teaches this feature. In particular, the Office Action stated that Smatch teaches a system where a modified compiler and scripts are used to perform static analysis. According to the Office Action, it would be obvious to incorporate Smatch's modified compiler and scripts into Granston's system.

However, even if Smatch's modified compiler arrangement were to be used in conjunction with Granston's wrapper program as urged in the Office Action, the resulting combination would still fail to meet the limitations of claim 11. The proposed Granston/Smatch combination would have a wrapper program that logged compiler options while calling Smatch's xgcc compiler. But the Granston/Smatch combination would not perform the claim 11 step of "using output from the monitoring program to run the build program with the static analysis tools substituted for the compilers so that the static analysis tools perform static error analysis on the source code files." At no point would the output of Granston's wrapper program (the log file) be used to run Granston's build program. The only use of the log file output that is disclosed in Granston is to allow users to ascertain which compiler options are enabled in a build environment (see column 1, lines 46-54 of Granston). There is nothing in Smatch that makes up for this deficiency in Granston. In particular, nothing in Smatch shows or suggests using any output of a program to run a build program, let alone the output of a program such as Granston's wrapper program. Smatch requires users to hard code a new compiler name xgcc into their makefile (see page 1 of SmUsing). Because the combination of Granston and Smatch does not meet all of the limitations of claim 11, claim 11 is patentable over

Granston and Smatch even if Granston and Smatch are combined as proposed in the Office Action. Claims 12-14 depend from claim 11 and are patentable because claim 11 is patentable.

Claims 15-20

Claim 15 is directed toward a method for using static analysis tools on a computer system to perform error analysis. The error analysis is performed on source code files that may be compiled on the computer system using compilers that are invoked using a build program during a build process. The method of claim 15 involves redefining operating system commands in the operating system. The build program is run on the computer system. As defined by claim 15, the redefined operating system commands cause the build program to invoke the static analysis tools in place of the compilers so that the error analysis on the source code files is performed.

In the Office Action, it was suggested that the modified compiler xgcc that is used to compile code in Smatch performs the claimed method step of "redefining operating system commands in the operating system." (Office Action, page 6, citing SmIR at page 1.)

However, at no point in the SmIR document or any of the other Smatch documents is there any suggestion of redefining operating system commands in an operating system. The SmIR

document describes how a modified compiler compiles source code and generates .c.sm files. A user then types a command "./checker_script.pl file.c.sm" to invoke a script. The script performs error analysis by analyzing the .c.sm files. At no point in the SmIR document or any of the other Smatch documents is there any suggestion to redefine operating system commands in an operating system.

Compilers are not operating systems. In particular, the xgcc compiler in Smatch is not an operating system nor is the xgcc compiler in Smatch part of an operating system. Because the compiler xgcc in Smatch is not an operating system or part of an operating system, use of the compiler xgcc in Smatch does not teach applicants' claimed step of redefining operating system commands as urged in the Office Action.

Applicants' approach for redefining operating system commands is illustrated by step 108 of FIG. 8 in applicants' patent application and is described on pages 31 and 32 of applicants' patent application. As shown in step 108 of FIG. 8, operating system commands that may be redefined include process creation and execution commands such as fork and exec. Nothing in Smatch makes any mention of fork or exec or any other operating system commands, let alone does Smatch teach that one should redefine operating system commands as described by

applicants in connection with FIG. 8 and as set forth in claim 15.

Because Smatch fails to disclose applicants' claimed method step of redefining operating system commands, claim 15 is not anticipated by Smatch. Claim 15 is therefore in condition for allowance. Claims 16-18 depend from claim 15 and are allowable because claim 15 is allowable.

Conclusion

As the foregoing demonstrates, claims 1-18 are in condition for allowance. This application is therefore in condition for allowance. Reconsideration of the application and allowance are respectfully requested.

Respectfully submitted,

Date: 3-16-07

/G. Victor Treyz/

G. Victor Treyz
Reg. No. 36,294
Attorney for Applicants
Customer No. 36532